# PENGARUH DESAIN TERHADAP PENERAPAN EFEKTIFITAS DATABASE MELALUI BEBERAPA CONTOH KASUS

**Agustinus Noertjahyana, Silvia Rostianingsih, Andreas Handojo**
Fakultas Teknologi Industri, Jurusan Teknik Informatika, Universitas Kristen Petra
e-mail: agust@petra.ac.id, silvia@petra.ac.id, , handojo@petra.ac.id

**ABSTRAK**: Dalam suatu sistem informasi, landasan yang utama adalah *database* dan implementasi prgoram. *Database* yang tidak efektif dan implementasi program yang tidak terstruktur dapat mempengaruhi performansi sistem informasi tersebut. Pengaruh desain terhadap *database* sangatlah besar, termasuk desain data, tipe data maupun relasinya. Pembuatan desain yang tidak dibangun dengan cermat dapat menyebabkan hilangnya data yang dibutuhkan, data yang tidak konsisten, redundansi data, proses *update* yang lambat dan banyak hal lain. Untuk menghindari hal tersebut, dibuatlah beberapa contoh kasus yang dapat menunjukkan betapa pentingnya desain sebelum pembuatan *database* yaitu pembuatan *logical data model*. Dari contoh kasus yang diberikan, dapat dilihat bahwa desain mempengaruhi *database* yang akan dibentuk.

**Kata kunci:** *database*, *logical data model.*

***ABSTRACT:*** *The main foundation of information system is database and programming. Inefficient database and unstructured programming can influence information system performance. Database design is very important, including the data design, data domain and relationship. Unstable design can cause data lost, inconsistent data, redundancy data, slow data updating and many more. A some presented in this paper study cases here show how important the step of design before the step of making database, which is called the design of logical data model. From these study cases here, we can see that design will influence the database.*

***Keywords:*** *database, logical data model.*

## PENDAHULUAN

Salah satu langkah dalam membangun suatu sistem informasi adalah melakukan perancangan *database*. *Database* merupakan jantung dari sistem informasi. Data harus tersedia ketika *user* ingin menggunakan, data juga harus akurat dan konsisten. Selain dari *requirement* tersebut, tujuan dari desain *database* adalah efisiensi penyimpanan data dan efisiensi pembacaan maupun *update* data.

*Database* merupakan suatu koleksi data. Efektifitas dari *database* harus dapat memenuhi kebutuhan (1) memastikan data agar dapat diakses oleh banyak *user* pada banyak aplikasi, (2) *maintain* data secara akurat dan konsisten, (3) memastikan data yang dibutuhkan baik sekarang maupun yang akan datang dapat tersedia, (4) *database* dapat memenuhi kebutuhan sesuai dengan pertumbuhan *user* dan (5) *database* dapat memenuhi kebutuhan pembacaan data tanpa memperdulikan bagaimana data secara fisik tersimpan [4].

Dari pengamatan yang dilakukan penulis, masih ada beberapa orang yang memiliki tidak mempertimbangkan efektifitas dalam mendesain *database*. Untuk menjelaskan lebih lanjut pentingnya efektifitas *database*, dibuatlah beberapa contoh kasus dalam desain *logical data model*.

## LOGICAL DATA MODEL

*Logical data model* merupakan pemodelan dari proses bisnis yang berfokus pada analisis data. *Logical data model* dibangun oleh tiga notasi yaitu entiti, atribut dan relasi. Entiti adalah tempat, obyek, kejadian maupun konsep pada lingkungan *user* dimana diperlukan *maintain* data pada organisasi tersebut. Atribut adalah karakteristik yang dimiliki tiap entiti. Relasi adalah hubungan asosiasi data antar entiti.

Beberapa hal yang perlu diperhatikan dalam pembuatan *logical data model* menurut Moss Larissa [2]:

a. Memeriksa definisi, semantik dan tipe data pada tiap entiti untuk mencari duplikasi obyek bisnis karena dapat tidak terlihat apabila nama yang digunakan berbeda.
b. Memastikan tiap data pada entiti bahwa hanya memiliki satu pengenal yang unik (*primary key*), dimana termasuk apabila ada data lama yang dihapus dari *database*.

c. Menggunakan aturan normalisasi untuk memastikan bahwa sebuah atribut hanya dimiliki oleh satu entiti saja.
d. Mengadopsi aturan bisnis dengan obyek pada dunia nyata. Aturan bisnis ini memperlihatkan relasi data antar entiti.

Beberapa hal yang perlu diperhatikan dalam pembuatan *logical data model* menurut Elmasri Ramez [1]:

a. Semantic atribut
   Bagaimana menggambarkan relasi yang dapat menggambarkan fakta yang ada.
b. Memperkecil terjadinya data redundansi
   Tujuan dalam pembuatan *database* adalah mengoptimalkan penyimpanan data.
c. Memperkecil terjadinya nilai null pada data
   Null value dapat menyebabkan penyimpanan data yang besar dan dapat terjadi kesalahpahaman dalam mengartikan suatu atribut. Null value dapat diinterpretasikan sebagai: (1) atribut ini tidak dimiliki oleh data tersebut, (2) nilai atribut tidak diketahui dan (3) nilai atribut diketahui tetapi belum dicatat.
d. Tiap entiti memiliki definisi/semantik yang jelas.

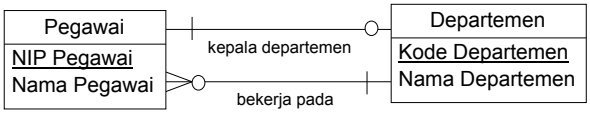
## PENERAPAN *LOGICAL DATA MODEL* DALAM CONTOH KASUS

Contoh kasus digambar menggunakan notasi *chicken feet* dengan *tools* Power Designer 6.1.

### Contoh kasus 1

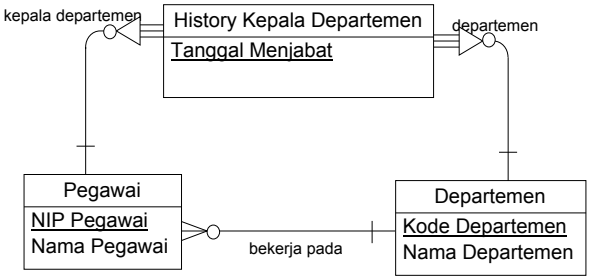
Gambar 1 menunjukkan relasi kepala departemen antara entiti Pegawai dengan Departemen adalah 1 : 1 yang berarti satu orang pegawai hanya dapat mengepalai satu departemen dan satu departemen hanya boleh dikepalai oleh satu orang pegawai.

Dilihat dari kenyataan yang terjadi, relasi tersebut adalah benar karena tidak mungkin pada satu waktu, ada lebih dari satu pegawai yang mengepalai suatu departemen dan begitu pula sebaliknya.



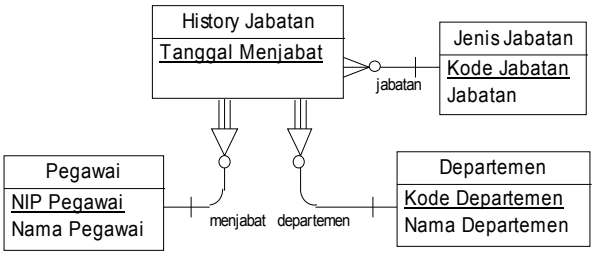

**Gambar 1. Relasi entiti Pegawai dan entiti Departemen**

Namun ternyata ketika terjadi pergantian kepala departemen, data kepala departemen yang lama sudah tidak dapat lagi diketahui. Dengan kata lain, *database* tidak menyediakan penyimpanan data masa lampau. Oleh karena itu, desain gambar 1 ditambahkan suatu entiti yang mencatat tanggal seorang pegawai menjabat suatu departemen, sehingga dapat ditelusuri siapa saja yang pernah menjabat menjadi kepala departemen suatu departemen (gambar 2).




**Gambar 2. Penambahan entiti History Kepala Departemen**

Apabila ruang lingkup ditambah dengan pencatatan setiap jabatan yang dipegang selama seorang pegawai, maka gambar 2 harus diubah lagi dengan memasukkan informasi pilihan jabatan yang mungkin dijabat seorang pegawai selain kepala departemen. Sebagai seorang pegawai pasti mempunyai sebuah jabatan, sehingga dari entiti history jabatan dapat diketahui departemen yang saat itu menaunginya. Oleh karena itu, relasi antara entiti Pegawai dengan entiti Departemen dapat dihilangkan.

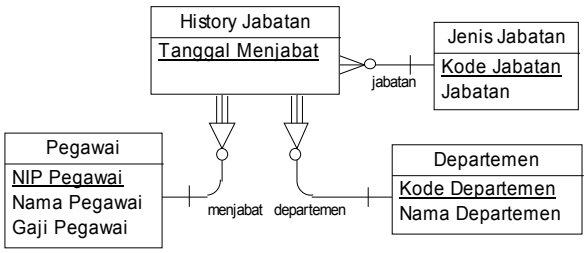



**Gambar 3. Perubahan entiti History Jabatan**

Moss poin (d) dan Elmasri poin (a) menyatakan bahwa pembuatan model harus disesuaikan dengan fakta. Contoh kasus 1 memperlihatkan bahwa dalam melakukan desain perlu memastikan data apa saja yang dibutuhkan baik sekarang maupun yang akan datang dapat tersedia. Pada gambar 1 terdapat permasalahan yaitu tidak menyediakan penyimpanan data masa lampau. Kemudian yang kedua adalah apakah *database* dapat memenuhi kebutuhan sesuai dengan pertumbuhan *user* sebagaimana yang digambarkan pada gambar 3 yaitu bahwa jenis jabatan dapat semakin beragam, oleh karena itu dibuat sebuah entiti tersendiri.

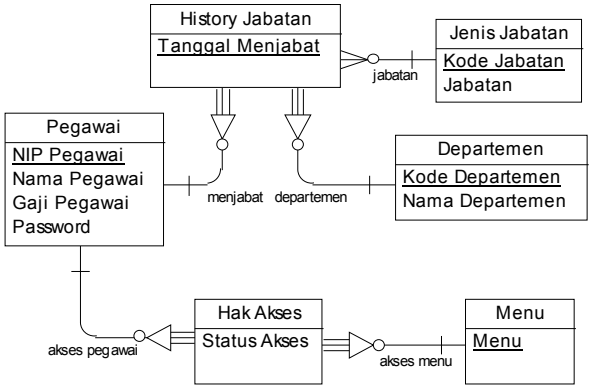
**Contoh kasus 2**

Pada suatu *database* yang diakses oleh banyak *user*, perlu diperhatikan data mana saja yang boleh diakses seorang pegawai, data mana yang tidak boleh diakses oleh seorang pegawai. Seperti yang dilihat pada gambar 4, setiap pegawai berhak melihat history jabatannya masing-masing maupun milik pegawai lain, namun setiap pegawai (kecuali bagian penggajian) hanya boleh melihat data gaji miliknya sendiri.




**Gambar 4. Pengaksesan history jabatan**

Untuk memastikan data agar dapat diakses oleh banyak *user* pada banyak aplikasi maka perlu diperhatikan pemilihan *database* yang digunakan. Apabila ternyata *database* yang digunakan tidak mendukung *management user*, maka perlu disediakan entiti Hak Akses dan entiti Menu (gambar 5).

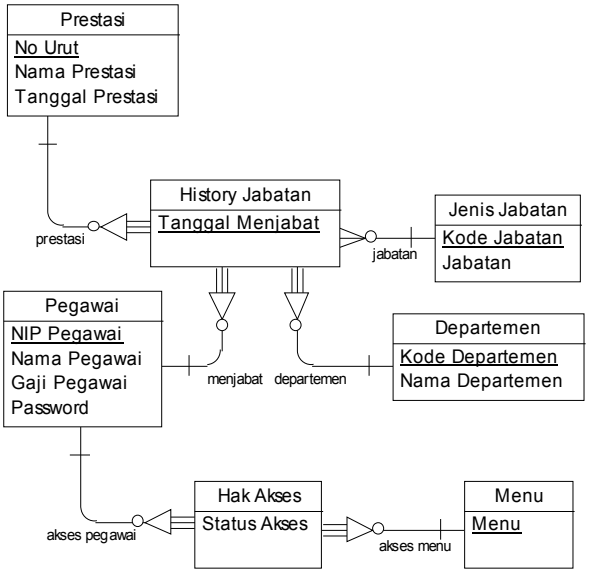



**Gambar 5. Penambahan hak akses menu**

Contoh kasus 2 memperlihatkan bahwa pemilihan *database* juga mempengaruhi desain *database*, antara lain adalah *management user* atau pemilihan tipe data, dimana tiap *database* menyediakan tipe data yang berbeda.

**Contoh kasus 3**

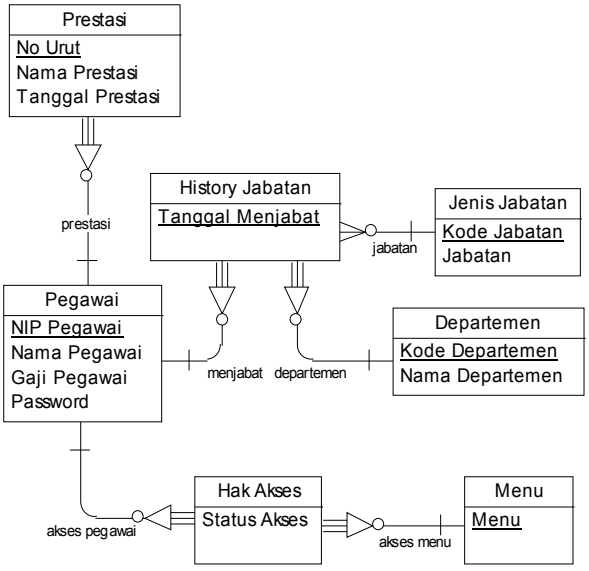
Seorang pegawai berprestasi akan dikaitkan dengan jabatan apa yang dimiliki pada saat itu, sehingga pada gambar 6 digambarkan bahwa entiti Prestasi direlasikan dengan entiti History Jabatan.




**Gambar 6. Entiti Prestasi berleasi dengan entiti Jabatan**

Ternyata prestasi tidak sepenuhnya melekat pada prestasi, namun sebenarnya lebih tepat apabila entiti Prestasi direlasikan dengan entiti Pegawai. Untuk mengetahui prestasi seorang pegawai didapat pada saat pegawai tersebut menjabat menjadi apa, dapat dilakukan proses *query*.

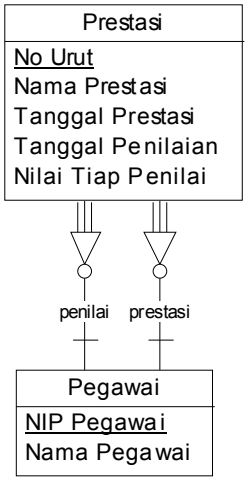
Contoh kasus 3 memperlihatkan bahwa ketika mengadopsi aturan bisnis dengan obyek pada dunia nyata harus dilakukan dengan cermat, sehingga relasi yang dibuat menjadi tepat (Moss poin (d) dan Elmasri poin (a) dan memastikan bahwa tiap definisi/semantik menempel pada data yang sesuai (Moss (a) dan Elmasri (d)).




**Gambar 7. Entiti Prestasi berelasi dengan entiti Pegawai**

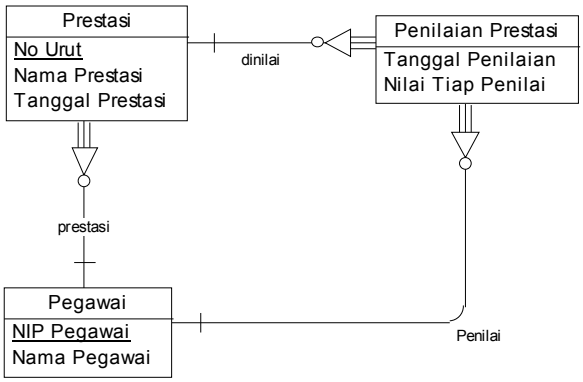
**Contoh kasus 4**

Normalisasi penting dilakukan untuk memastikan bahwa sebuah atribut hanya dimiliki oleh satu entiti saja sehingga *maintain* data dapat dilakukan secara akurat dan konsisten. Data yang tidak ternormalisasi juga menyebabkan lambatnya proses *update*. Setiap prestasi seorang pegawai dinilai oleh beberapa penilai. Jumlah penilai tergantung pada jabatan yang dimiliki oleh pegawai yang berprestasi tersebut. Dilihat dari gambar 8, ternyata terjadi redundansi data, dimana terjadi perulangan data atribut No Urut, Nama Prestasi dan Tanggal Prestasi tiap ada nilai dari seorang penilai.




**Gambar 8. Redundansi data pada entiti Prestasi**

Gambar 8 dapat diubah menjadi gambar 9 untuk menghilangkan redundansi data.




**Gambar 9. Pemisahan redundansi data pada entiti Penilaian Prestasi**

Contoh kasus 4 memperlihatkan bahwa dalam melakukan desain perlu memperhatikan aturan normalisasi (Moss poin (c) dan Elmasri poin (b)), sehingga konsistensi data dapat terjamin. Proses *update* data akan menjadi lebih sulit apabila dilakukan pada desain *database* seperti gambar 8. Contoh data pada tabel Prestasi dari gambar 8 dapat dilihat pada tabel 1.

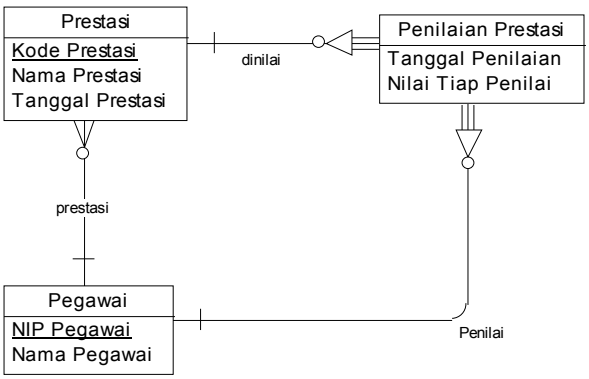
**Tabel 1. Contoh data tabel Prestasi**

| NIP Pegawai | No Urut | Nama Prestasi | NIP Penilai |
|---|---|---|---|
| 01043 | 1 | Technology Award 2003 | 87021 |
| 01043 | 1 | Technology Award 2003 | 99073 |
| 01043 | 1 | Technology Award 2003 | 96033 |

Apabila terjadi kesalahan dalam pemasukan data atribut Nama Prestasi dari ’Technology Award 2003’ menjadi ’Technology Award 2004’, maka *update* harus dilakukan pada tiga *record*) dalam tabel Prestasi. Berbeda dengan data yang tersimpan dengan desain pada gambar 9, proses *update* hanya perlu dilakukan pada satu *record* saja.

**Contoh kasus 5**

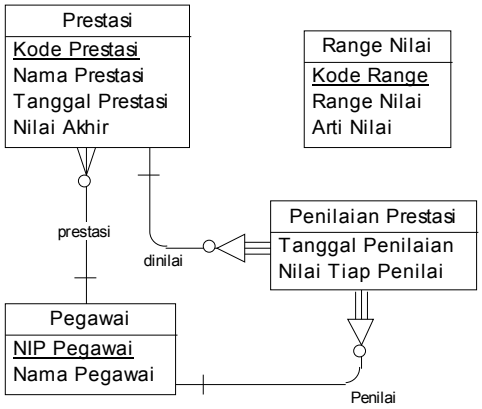
Hal lain yang perlu diperhatikan adalah pemilihan atribut sebagai *primary key* (Moss poin (b)). Antara gambar 9 dengan gambar 10 terdapat perbedaan *primary key* untuk entiti Prestasi. Gambar 9 menghasilkan tabel Prestasi dengan *primary key* NIP pegawai dan No Urut. Sedangkan gambar 10 menghasilkan tabel prestasi dengan *primary key* Kode Prestasi. Pemilihan desain tergantung daripada desainer *database*. Apabila dicermati, Kode Prestasi pada tabel prestasi dapat dihilangkan karena dengan NIP Pegawai dan No Urut tidak menyebabkan kesulitan proses *update* data pada tabel Prestasi.




**Gambar 10. Primay Key Tabel Prestasi adalah Kode Prestasi**

**Contoh kasus 6**

Pendapat yang mengatakan bahwa semua tabel dalam suatu sistem informasi harus memiliki relasi adalah tidak sepenuhnya benar. Ada beberapa tabel yang memang harus berdiri sendiri karena tabel tersebut hanya sebagai tabel bantu.

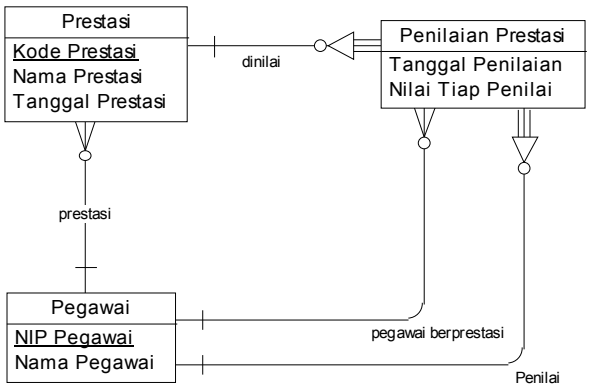



**Gambar 11. Entiti Range Nilai tidak harus berelasi dengan entiti lain**

Hasil dari penilaian prestasi dari tiap penilai akan dikalkulasi sehingga menghasilkan nilai akhir (disimpan pada atribut Nilai Akhir pada entiti Prestasi). Pada gambar 11 terdapat entiti Range Nilai yang menyimpan informasi 'arti' tiap nilai akhir pada tiap prestasi. Entiti Range Nilai tidak perlu direlasikan dengan entiti Prestasi karena yang disimpan pada entiti Range Nilai merupakan informasi *range* nilai dan arti nilai, bukan menyimpan tiap nilai dan arti nilai. Sehingga entiti Range Nilai cukup berdiri sendiri.

**Contoh kasus 7**

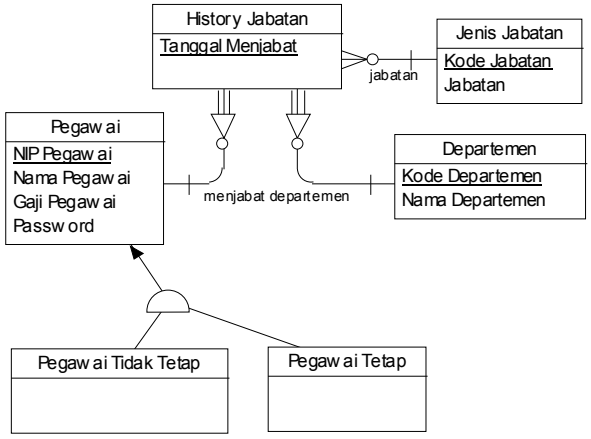
Pembuatan relasi yang berlebihan juga menyebabkan redundasi data (Elmasri poin (b)). Gambar 12 memperlihatkan bahwa ada redundasi relasi dimana pada tabel Penilaian Prestasi disimpan atribut NIP Pegawai, padahal dari relasi dinilai (antara entiti Prestasi dengan entiti Penilaian Prestasi) dapat dicari pula NIP Pegawai yang dinilai prestasinya dengan menggunakan proses *query*. Sehingga relasi pegawai berprestasi harus dihilangkan untuk menghilangkan redundansi data.




**Gambar 12. Redundasi relasi pada relasi pegawai berprestasi**

**Contoh kasus 8**

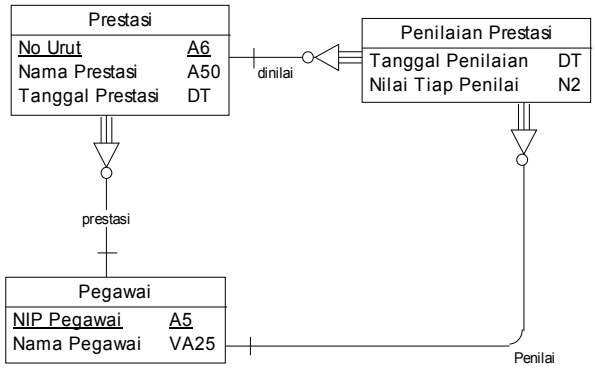
Pemilihan bentuk desain dapat juga mempengaruhi penyimpanan nilai null pada data (Elmasri poin (c)). Pada gambar 13 terlihat bahwa Pegawai dibedakan menjadi dua jenis yaitu pegawai tidak tetap dan pegawai tetap. pegawai tidak tetap tidak akan memiliki history jabatan seperti pegawai tetap. Oleh karena itu, relasi menjabat seharusnya tidak direlasikan antara entiti Pegawai dengan entiti History Jabatan melainkan direlasikan antara entiti Pegawai Tetap dengan entiti History Jabatan.




**Gambar 13. Redundasi relasi pada relasi pegawai berprestasi**

**Contoh kasus 9**

Pemilihan tipe data juga merupakan hal yang penting dalam melakukan desain. Salah satu contohnya adalah tipe data antara *character* dengan *variable character*. *Character*(n) berarti menyimpan data sejumlah n karakter. *Variable character*(n) berarti menyimpan data sejumlah karakter yang dimasukkan. Oleh karena itu, untuk jumlah karakter data yang sama, digunakan tipe data *character*, seperti atribut NIP Pegawai. Sedangkan untuk atribut Nama Pegawai digunakan tipe data *variable character*.




**Gambar 14. Pemilihan tipe data**

Penggunaan tipe data pada gambar 14 untuk atribut Nama Prestasi pada entiti Prestasi menyebabkan penggunaan *space* yang tidak perlu setiap kali terjadi penambahan data dari tabel Prestasi. Tipe data atribut tersebut harus diubah dari *Variable Character*(50) menjadi *Character*(50).

## KESIMPULAN

Berdasarkan dari pembahasan sebelumnya, maka dapat diambil kesimpulan yaitu:

1. Kesalahan yang sering terjadi dalam melakukan desain adalah (1) tidak mempersiapkan desain yang dapat digunakan pada perkembangan sistem di masa yang akan datang, (2) pembuatan relasi yang salah, (3) pembuatan relasi yang redundansi, (4) adanya redundansi data, (5) pemilihan primary key untuk suatu tabel dan (6) pemilihan tipe data.
2. Dalam mendesain *logical data model* harus memperhatikan *database* yang akan digunakan, proses bisnis pada sistem dan mempersiapkan desain yang dapat digunakan pada perkembangan sistem di masa yang akan datang.

## DAFTAR PUSTAKA

1. Elmasri Ramez, Navathe Shamkant B., *Fundamentals of Database Systems*. 3rd edition. 2000.
2. Moss Larissa, Hoberman Steve, *The Importance of Data Modelling as a Foundation for Business Insight,* 2004.
3. Hoffer, J.A., George J.F. and Valacich J.S., *Modern System Analysis and Design*, Second Edition. Addison Wesley Longman Inc. USA, 1999.
4. Whitten J. L., *System Analysis and Design Methods*. 5th edition. McGraw-Hill, 2001.